Project Title

Model Order Selection and Cross-Validation for Gaussian Mixture Models: A Comprehensive Study on 2D Data Synthesis

Project by
Shiva Kumar Dande

Project Completion Date:

March 2023

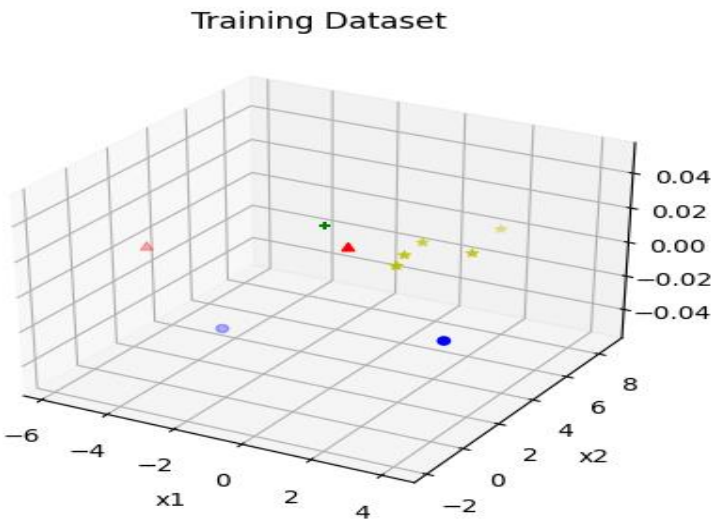
Problem Statement

Conduct the following model order selection exercise using 10-fold cross-validation procedure and report your procedure and results in a comprehensive, convincing, and rigorous fashion:

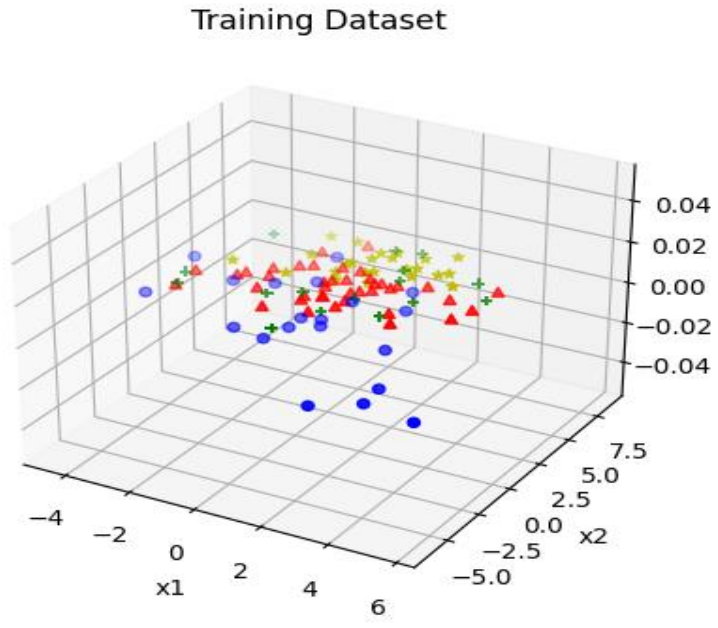
1. Select a Gaussian Mixture Model as the true probability density function for 2-dimensional real-valued data synthesis. This GMM will have 4 components with different mean vectors, different covariance matrices, and different probability for each Gaussian to be selected as the generator for each sample. Specify the true GMM that generates data.
2. Generate multiple data sets with independent identically distributed samples using this true GMM; these datasets will have respectively 10, 100, 1000, 10000 samples.
3. For each data set, using maximum likelihood parameter estimation principle (e.g. with the EM algorithm), within the framework of K-fold (e.g., 10-fold) cross-validation, evaluate GMMs with different model orders; specifically evaluate candidate GMMs with 1, 2, 3, 4, 5, 6 Gaussian components. Note that both model parameter estimation and validation performance measures to be used is log-likelihood of data.
4. Repeat the experiment multiple times (e.g., at least 30 times) and report your results, indicating the rate at which each of the the six GMM orders get selected for each of the datasets you produced. Develop a good way to describe and summarize your experiment results in the form of tables/figures.

A Gaussian Mixture Model is employed to produce 2D data featuring four unique classes, each with varying class priors. This setup leads to the generation of diverse datasets consisting of 10, 100, 1,000, and 10,000 samples. The visualization of the data distribution for these datasets can be observed below.

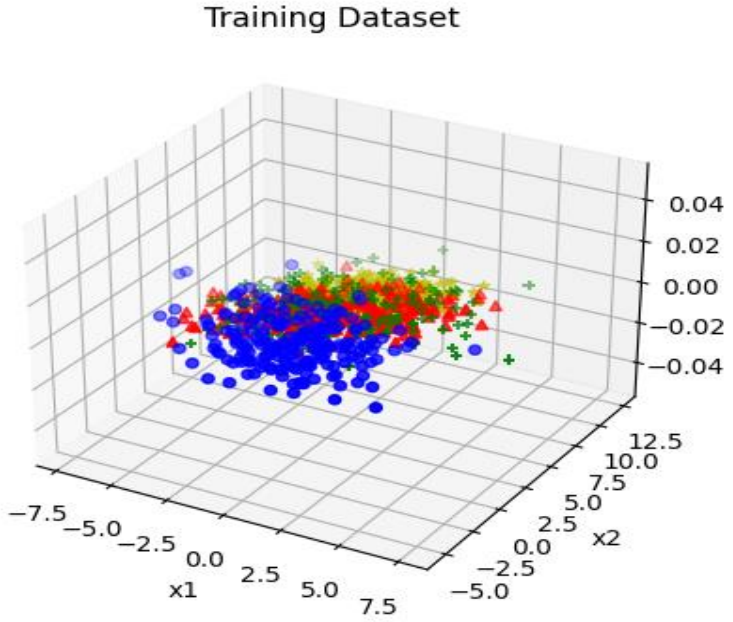
Data set with 10 points.



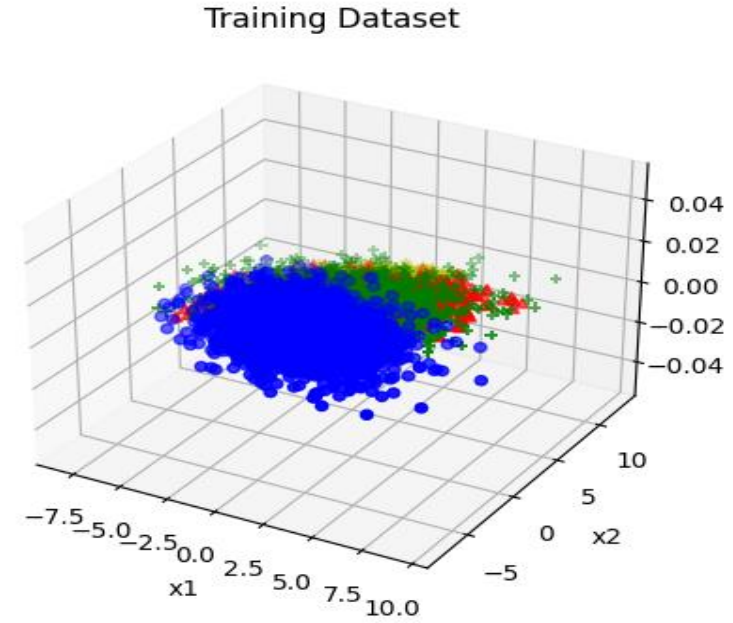
Data set with 100 points.



Data set with 1000 points.



Data set with 10000 points.



Priors, Mean and Co-variance Values used for the above data is shown below.

```
    gmmParameters['priors'] = [0.2, 0.3, 0.35, 0.15] # priors should be a row
vector
    #gmmParameters['meanVectors'] = np.array([[0, 0], [0, 40], [40, 0], [40,
40]])
    gmmParameters['meanVectors'] = np.array([[0, 0], [0, 4], [0, 3], [0, 4]])
    gmmParameters['covMatrices'] = np.zeros((4, 2, 2))
    gmmParameters['covMatrices'][0,:,:] = np.array([[2, -6], [-6, 2]])
    gmmParameters['covMatrices'][1,:,:] = np.array([[4, 2], [2, 4]])
    gmmParameters['covMatrices'][2,:,:] = np.array([[2, 1], [1, 2]])
    gmmParameters['covMatrices'][3,:,:] = np.array([[7, 1], [1, 7]])
```

Gaussian Mixture Model Implementation

To apply different Gaussian Mixture Models in Python, we use the GaussianMixture class from the Scikit-Learn library. This class employs the Expectation-Maximization (EM) algorithm, which iteratively calculates the optimal number of Gaussian components to fit the data distribution. The Expectation step estimates missing data values based on the current parameter estimates, while the Maximization step updates the Gaussian components' parameters (weights, means, and covariances) to identify the best solution.

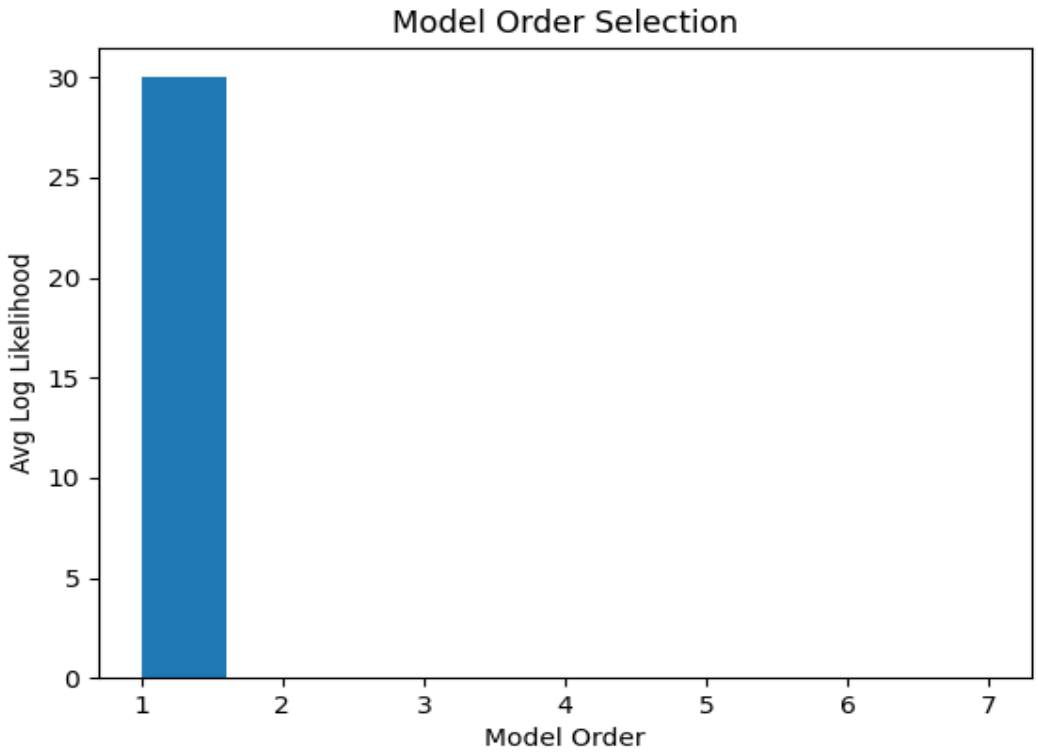
Three crucial hyperparameters for the GaussianMixture object are max_iter (maximum number of iterations), n_init (number of initializations), and tol (tolerance). Tolerance primarily serves as the convergence threshold, ending the iterations when the average gain falls below this value. Max_iter acts as a stopping criterion to prevent excessively long training times. The model is initialized multiple times randomly using n_init, and the best results are retained.

Model Order Selection with Cross Validation

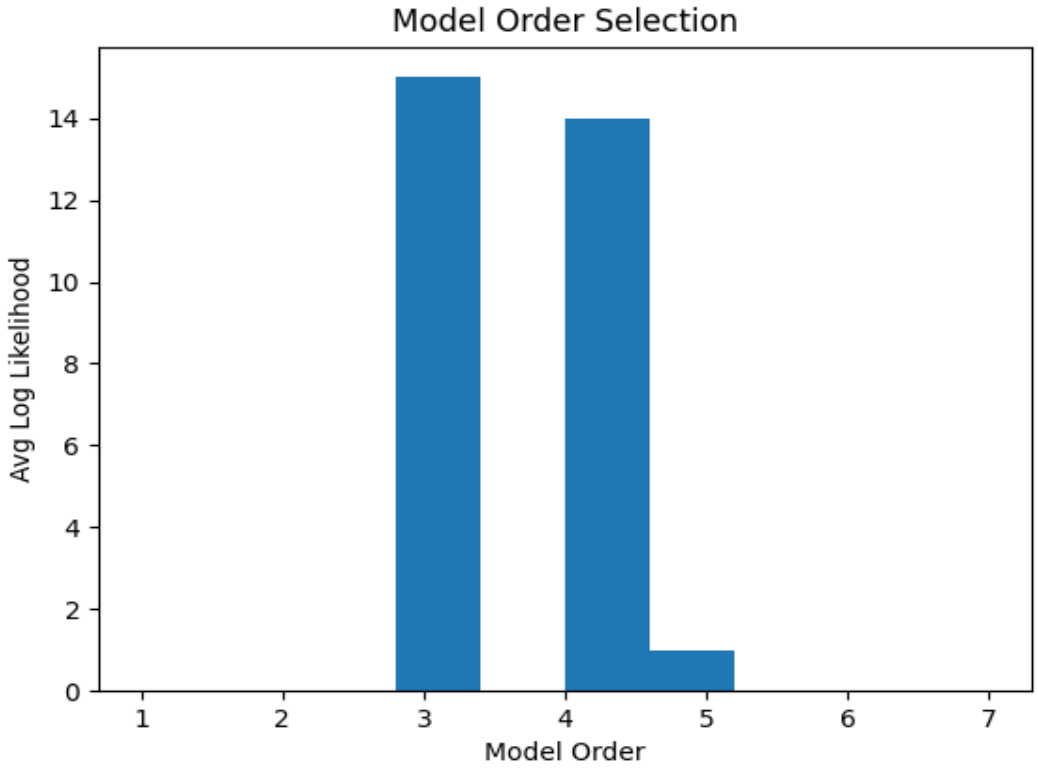
To explore different model orders, 10-Fold Cross-Validation is employed similarly to the previous problem. The model order or number of Gaussian components is varied from 1 to 6 to examine how often each order is chosen for a specific dataset. For each tested model order, the dataset is divided into 10 folds, with one-fold sequentially reserved as the validation set and the rest used for model fitting (training). During testing, the score() function from the Gaussian Mixture class is invoked with the validation set as input, calculating the log-likelihood of data samples. After completing the cross-validation process, the average of the ten scores is computed and compared to determine the model order with the highest score. The training is repeated approximately 50 times to assess the frequency of selected model orders using histogram graphs.

Output:

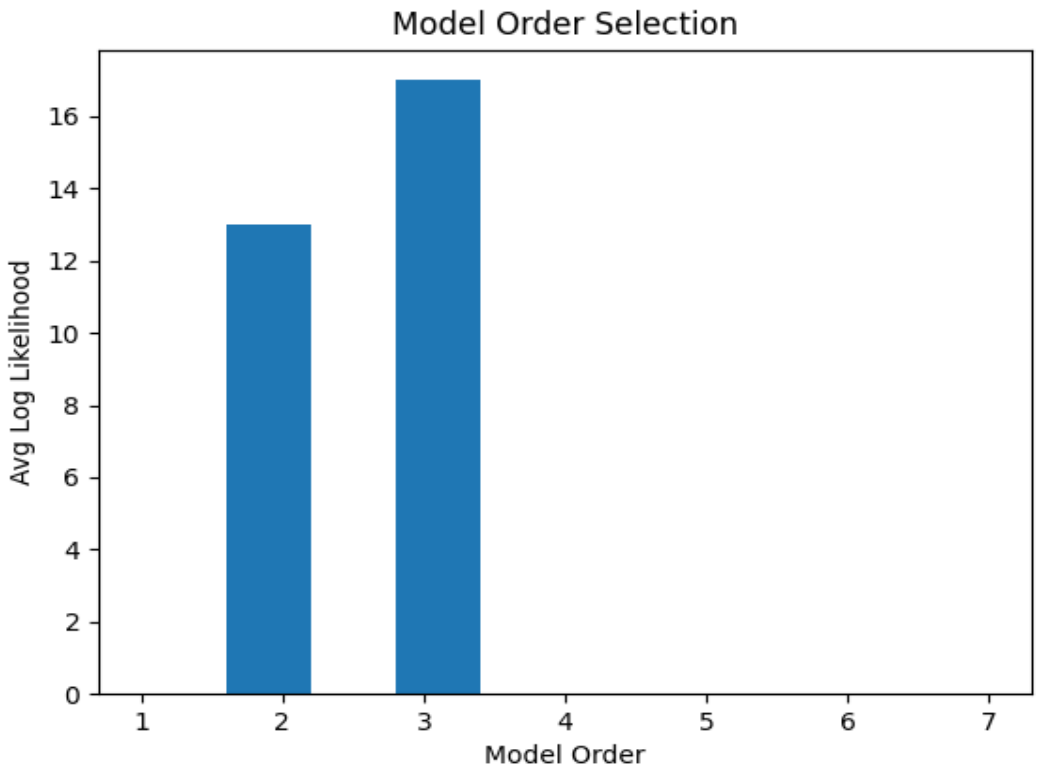
Data set with 10 points



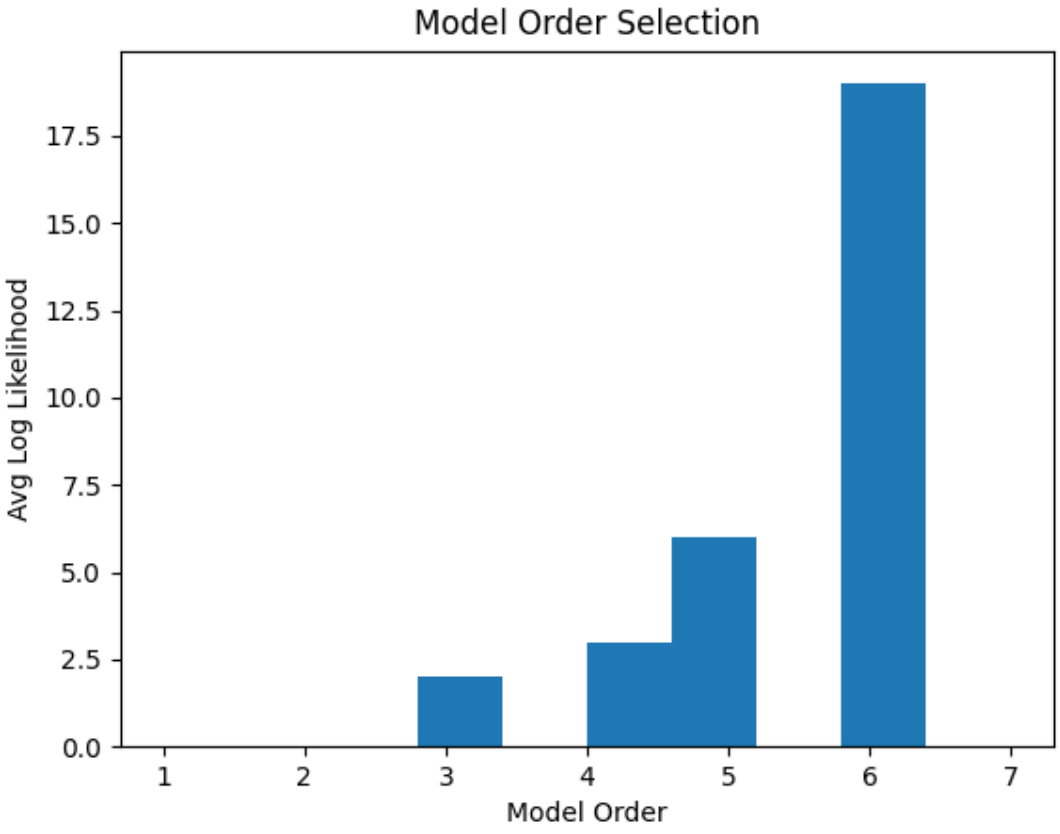
Data set with 100 points



Data set with 1000 points



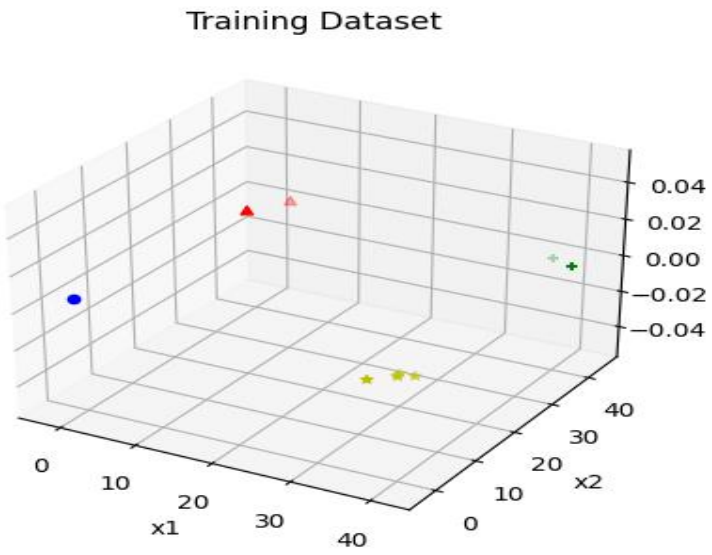
Data set with 10000 points.



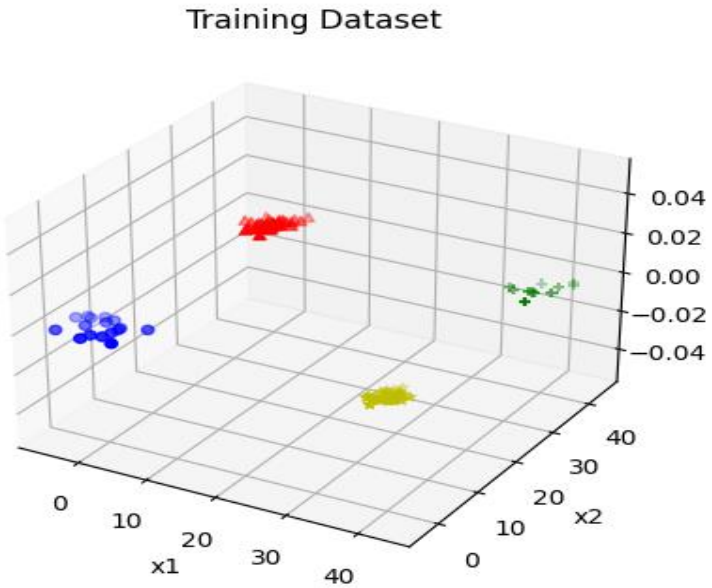
It can be seen that when working with 10 samples, the model consistently fits all data points within a single Gaussian component. With 100 samples, the model occasionally chooses a higher number of components but primarily still opts for 1 or 2 components. However, with 1,000 and 10,000 samples, the model more effectively recognizes the distribution and selects higher model orders like 4, 5, and 6 with greater frequency. Due to the overlapping nature of the different classes in the data distribution, the model struggles to accurately identify the correct number of components.

Experiments were also conducted on a data distribution featuring non-overlapping clusters, as displayed below:

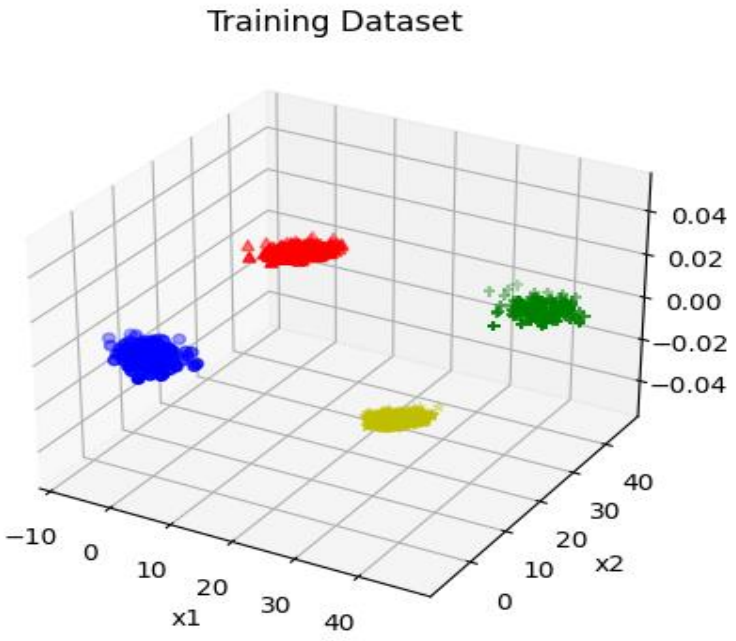
Data Set with 10 points.



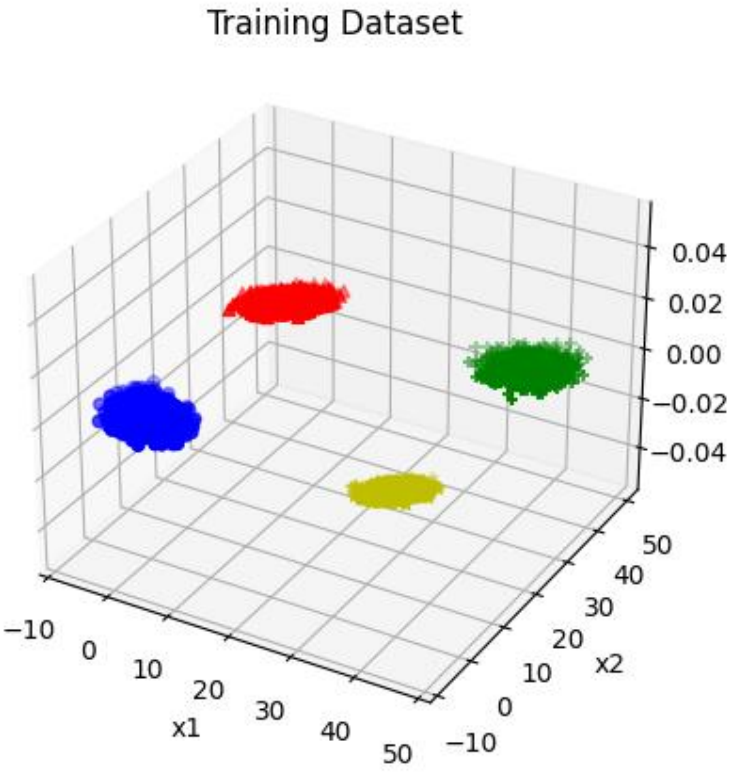
Data set with 100 points.



Data set with 1000 points.



Data set with 10000 points

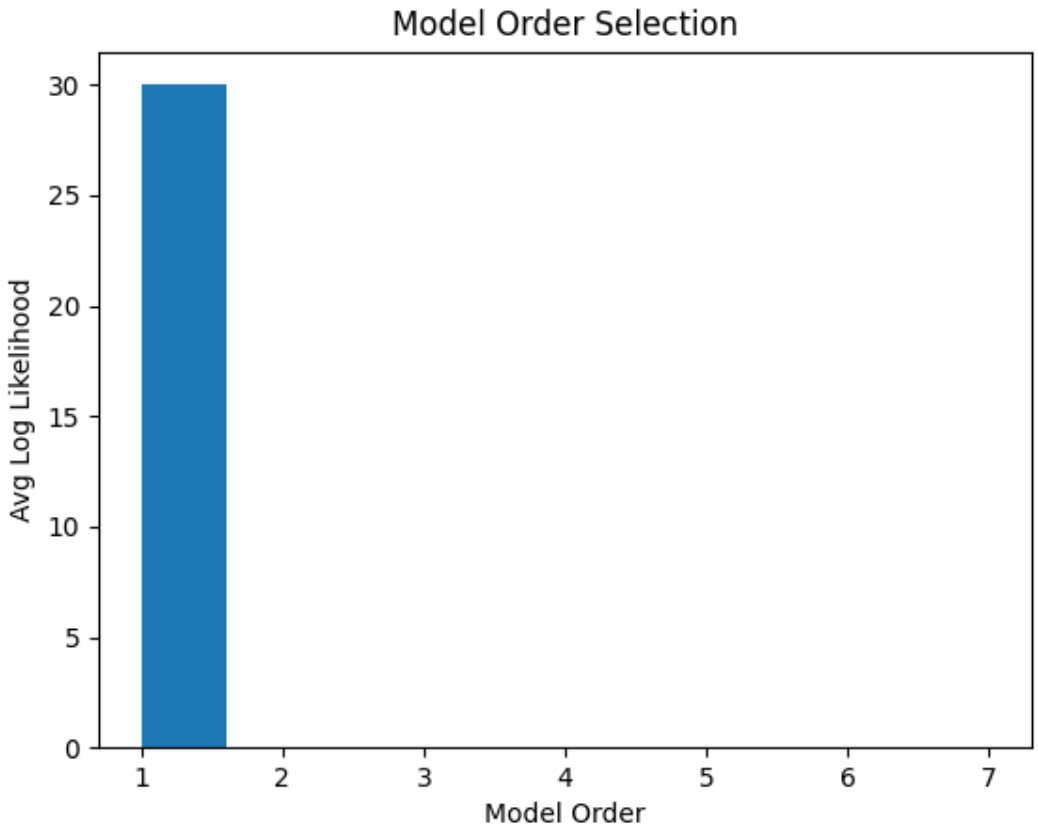


Priors, Mean and Co-variance Values used for the above data is shown below.

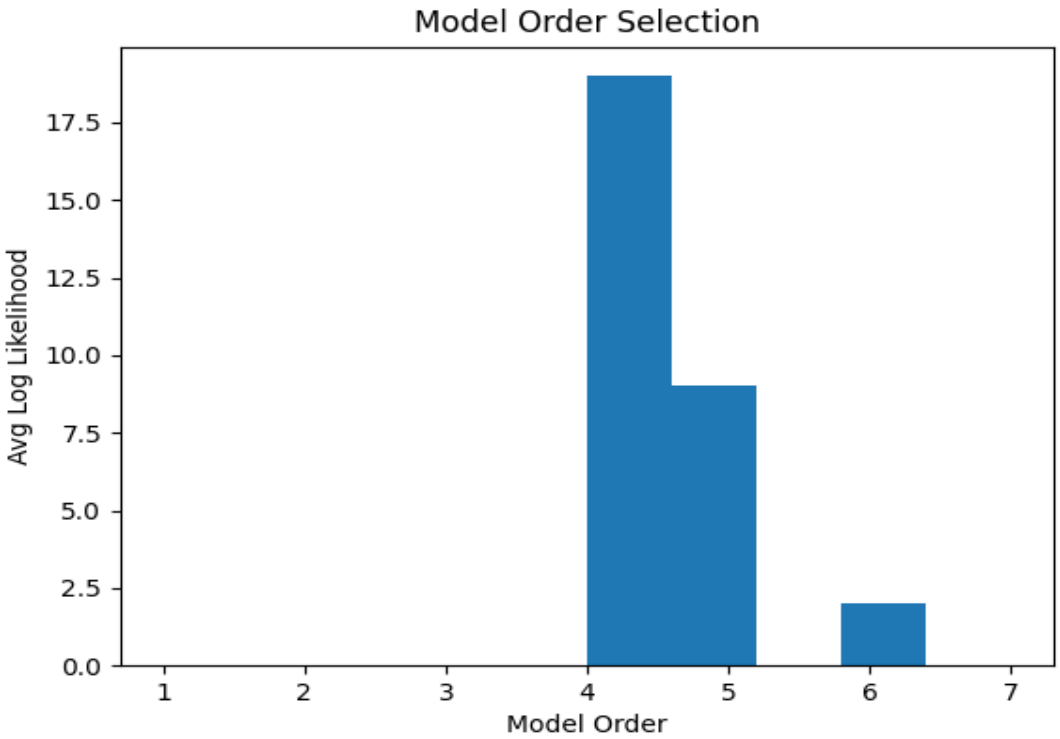
```
    gmmParameters['priors'] = [0.2, 0.3, 0.35, 0.15] # priors should be a row
vector
    gmmParameters['meanVectors'] = np.array([[0, 0], [0, 40], [40, 0], [40,
40]])
    #gmmParameters['meanVectors'] = np.array([[0, 0], [0, 4], [0, 3], [0, 4]])
    gmmParameters['covMatrices'] = np.zeros((4, 2, 2))
    gmmParameters['covMatrices'][0,:,:] = np.array([[2, -6], [-6, 2]])
    gmmParameters['covMatrices'][1,:,:] = np.array([[4, 2], [2, 4]])
    gmmParameters['covMatrices'][2,:,:] = np.array([[2, 1], [1, 2]])
    gmmParameters['covMatrices'][3,:,:] = np.array([[7, 1], [1, 7]])
```

Outputs:

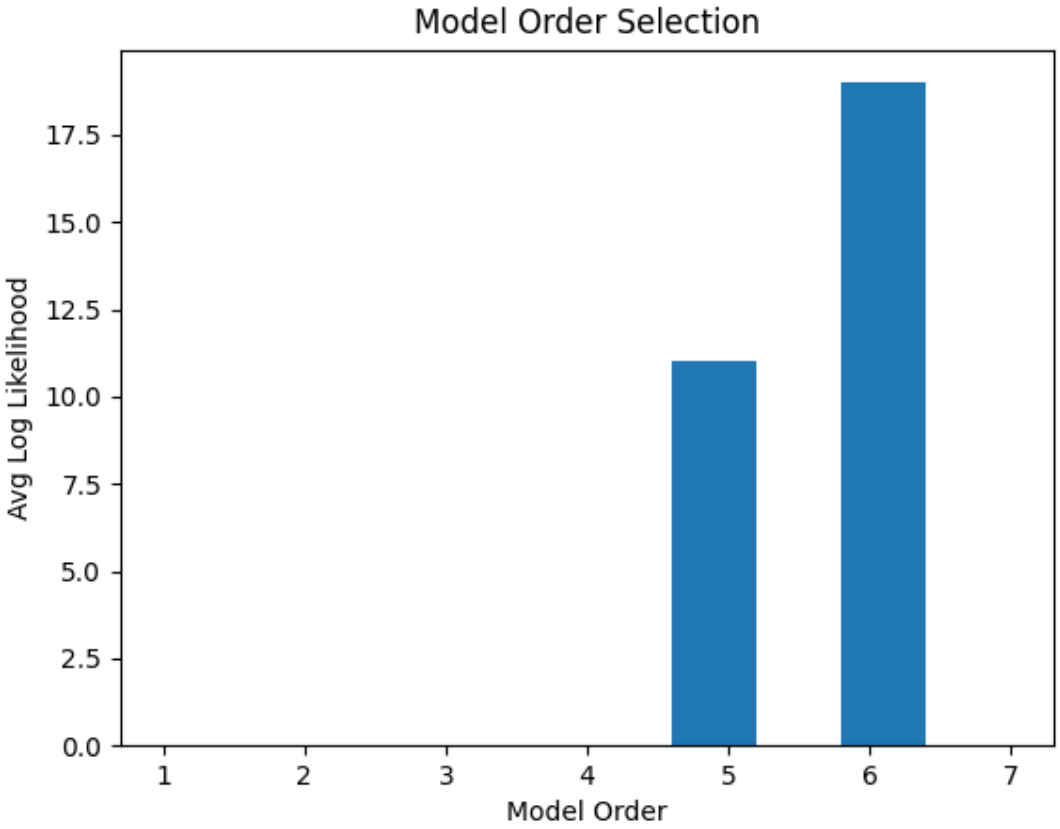
Data set with 10 Points.



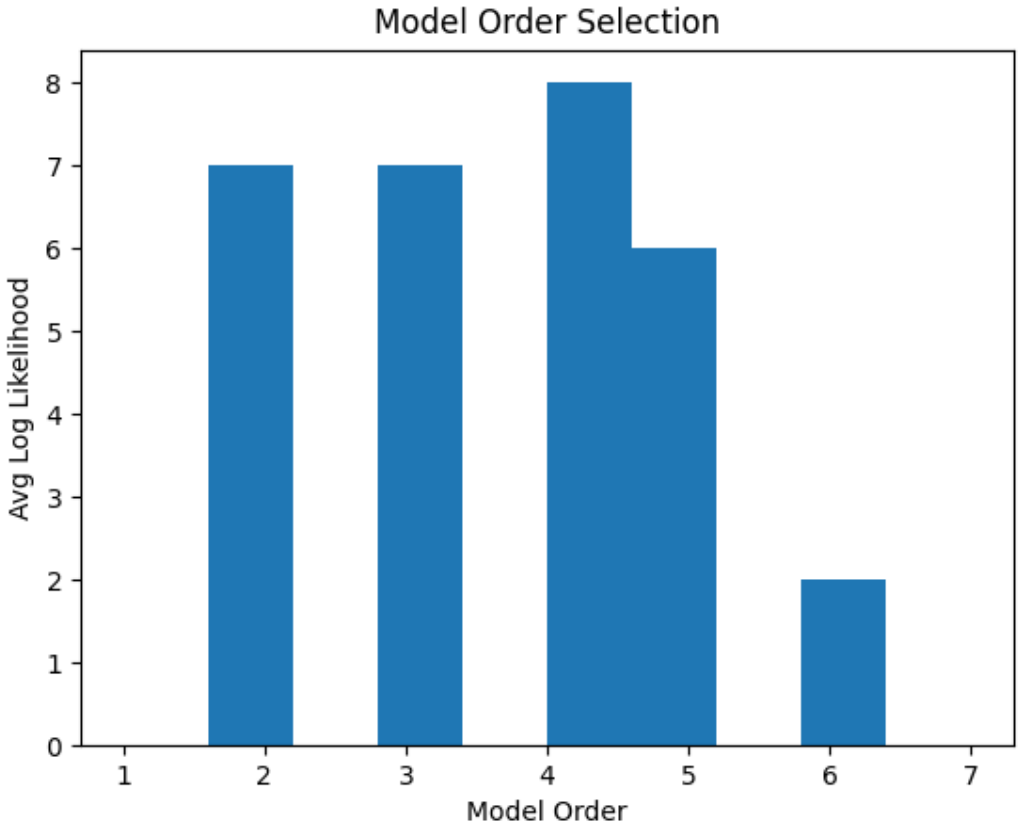
Data set with 100 Points.



Data set with 1000 points.



Data set with 10000 points.



In this scenario, the model still tends to fit 10 samples into a single Gaussian component. However, with 100 samples, it mostly succeeds in accurately identifying the correct number of components, and with 1,000 samples, it consistently yields the correct result. For 10,000 samples, each training routine takes a substantial amount of time, which is why only 10 routines were performed. In some instances, the model did not converge and required more than 3,000 iterations, making the process extremely time-consuming. In these cases, the max_iter hyperparameter played a role in stopping the process, resulting in the incorrect identification of 5 as the number of components.

In conclusion, the limitations of GMM include the expectation of non-overlapping data clusters for accurate classification and the need for a large number of iterations to achieve convergence. Additionally, hyperparameters demand careful fine-tuning to yield effective results.

Appendix:

Data Generation:

```
import numpy as np
import pandas as pd
from numpy.linalg import inv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def dataset_creation():
    Ntrain = [10, 100, 1000, 10000]
    for Nsamples in Ntrain:
        data, labels = generateData(Nsamples)
#         stacking the data (3 X 100) and labels (1 X 100) to create a data
set of size 100 X 4
        data_set = pd.DataFrame(np.transpose(np.vstack((data, labels))))
        plot3(data, labels, 'T')
#         stroing the egenrated data set in a .csv file
        filename = 'GMM_training_dataset' + str(Nsamples) + '.csv'
        data_set.to_csv(filename, index = False)

# #     creating validation data sets
#     NValidation = 100000
#     data, labels = generateData(NValidation)
#     plot3(data, labels, 'V')
#     data_set = pd.DataFrame(np.transpose(np.vstack((data, labels))))
#     filename = 'validation_dataset' + str(NValidation) +'.csv'
#     data_set.to_csv(filename, index = False)

def generateData(N):
    gmmParameters = {}
#     given that priors are uniform so using 1/4 for each
    gmmParameters['priors'] = [0.2, 0.3, 0.35, 0.15] # priors should be a row
vector
    gmmParameters['meanVectors'] = np.array([[0, 0], [0, 30], [30, 0], [30,
30]])
    #gmmParameters['meanVectors'] = np.array([[0, 0], [0, 4], [0, 3], [0, 4]])
    gmmParameters['covMatrices'] = np.zeros((4, 2, 2))
    gmmParameters['covMatrices'][0,:,:] = np.array([[1, -3], [-3, 1]])
    gmmParameters['covMatrices'][1,:,:] = np.array([[8, 4], [4, 8]])
    gmmParameters['covMatrices'][2,:,:] = np.array([[6, 3], [3, 6]])
    gmmParameters['covMatrices'][3,:,:] = np.array([[7, 1], [1, 7]])
    x, labels = generateDataFromGMM(N,gmmParameters)
    return x, labels

def generateDataFromGMM(N,gmmParameters):
#    Generates N vector samples from the specified mixture of Gaussians
#    Returns samples and their component labels
```

```
#    Data dimensionality is determined by the size of mu/Sigma parameters
    priors = gmmParameters['priors'] # priors should be a row vector
    meanVectors = gmmParameters['meanVectors']
    covMatrices = gmmParameters['covMatrices']
    n = meanVectors.shape[1] # Data dimensionality
    C = len(priors) # Number of components
    x = np.zeros((n,N))
    labels = np.zeros((1,N))
    # Decide randomly which samples will come from each component
    u = np.random.random((1,N))
    thresholds = np.zeros((1,C+1))
    thresholds[:,0:C] = np.cumsum(priors)
    thresholds[:,C] = 1
    for l in range(C):
        indl = np.where(u <= float(thresholds[:,l]))
        Nl = len(indl[1])
        labels[indl] = (l)*1
        u[indl] = 1.1
        x[:,indl[1]] =
np.transpose(np.random.multivariate_normal(meanVectors[l,:],
covMatrices[l,:,:], Nl))
    labels = np.squeeze(labels)
    return x,labels

def plot3(data, labels, dtype):
#   from matplotlib import pyplot
#   import pylab
#   from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(data[0,labels == 0], data[1,labels == 0], marker='o',
color='b')
    ax.scatter(data[0,labels == 1], data[1,labels == 1], marker='^',
color='r')
    ax.scatter(data[0,labels == 2], data[1,labels == 2], marker='*',
color='y')
    ax.scatter(data[0,labels == 3], data[1,labels == 3], marker='+',
color='g')
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    if (dtype == 'T'):
        ax.set_title('Training Dataset')
    else:
        ax.set_title('Validation Dataset')
    plt.show()

dataset_creation()
```

Gaussian Mixture:

```
import numpy as np
import pandas as pd
import collections
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import KFold
from random import seed
from random import randrange
from scipy.stats import multivariate_normal

# Load your dataset into a Pandas dataframe or NumPy array.
data = pd.read_csv('GMM_training_dataset10000.csv')
input_features = data.iloc[:, : -1].to_numpy()
labels = data.iloc[:, -1].to_numpy()

# Define the number of folds for cross-validation
K = 10

# Initialize KFold
kf = KFold(n_splits=K)

# Define the range of Gaussian components to evaluate
num_components = [1, 2, 3, 4, 5, 6]

order = []

# Initialize a list to store the average log-likelihoods for each GMM
avg_log_likelihoods = []

for routine in range(30):
    avg_log_likelihoods = []

    for num in num_components:
        # Initialize a list to store the log-likelihoods for each fold
        log_likelihoods = []
        for train_index, val_index in kf.split(input_features):
            # Split the data into training and validation sets
            X_train, X_val = input_features[train_index],
input_features[val_index]
            # Fit the GMM using the EM algorithm
            gmm = GaussianMixture(n_components=num, init_params = 'random',
max_iter = 3000, tol = 1e-5, n_init = 2)
            gmm.fit(X_train)

            # Calculate the log-likelihood of the validation set
            log_likelihood = gmm.score(X_val)
```

```
            # Append the log-likelihood to the list for this fold
            log_likelihoods.append(log_likelihood)

        # Calculate the average log-likelihood across all K folds
        avg_log_likelihood = np.mean(log_likelihoods)
        #print(log_likelihoods)
        print(avg_log_likelihood)

        # Append the average log-likelihood to the list for all GMMs
        avg_log_likelihoods.append(avg_log_likelihood)

    print(avg_log_likelihoods)
    # Identify model order with maximum score
    order.append(np.argmax(avg_log_likelihoods) + 1)
    #print(f'order is : {order}')

#print(order)
best_num_components = num_components[np.argmax(avg_log_likelihoods)]
# best_gmm = GaussianMixture(n_components=best_num_components,
random_state=42)
# best_gmm.fit(input_features)

print(f"Best number of Gaussian components: {best_num_components}")

 #Plot a histogram showing frequency of model orders selected
plt.hist(order, range = (1, 7))
plt.xlabel("Model Order")
plt.ylabel("Avg Log Likelihood")
plt.title("Model Order Selection")
plt.show()
```